Supporting Knowledge Acceleration for Programming from a Sensemaking Perspective

Michael Xieyang Liu
HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
xieyangl@cs.cmu.edu

Angelina Zhou
HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
ajzhou@andrew.cmu.edu

Shaun Burley
HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
me@shaunburley.com

Aniket Kittur
HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
nkittur@cs.cmu.edu

Emily Deng
HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
edeng@cs.cmu.edu

Brad A. Myers
HCI Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
bam@cs.cmu.edu

Sensemaking Workshop: CHI 2018
April 21, 2018, Montreal, QC, Canada.

Abstract
Programmers spend a significant proportion of their time searching for and making sense of complex information. However, they often lack effective tools to help them make sense of the information, turn it into knowledge, or share it with their respective communities. In this position paper, we aim to help programmers collect, navigate, and organize knowledge to meet their goals while capturing this knowledge and making it useful for later programmers with similar needs. We describe barriers and challenges to creating this sustainable cycle, and we explore the design space and opportunities for effective tools and systems.

Author Keywords
Empirical Studies of Programmers; Development Tools; Sensemaking; Collaboration; Knowledge Transfer; Exploratory Search.

ACM Classification Keywords
H.5.2 [User interfaces (User-centered design)]: Miscellaneous

Introduction
Programming is highly cognitively demanding, requiring programmers to perform many activities at the same time, for which there is little direct support. For example, programmers often must understand existing code written

by others, determine how to write new code based on a large number of constraints and requirements, identify relationships and design rationale of code in order to make changes that work correctly, select among a set of application programming interfaces (APIs, also called libraries, toolkits or software development kits – SDKs), each having different properties and constraints. In fact, researchers have identified hundreds of questions that programmers ask and want to find answers for [28, 15], from basic questions like *"what is the syntax for writing a while loop in Python"* to more strategical and high-level ones like *"how to progressively adopt React.js (a JavaScript library developed by Facebook) in my existing project"*.

All of these cognitive tasks can be classified as attempts by the programmers to gain knowledge about their code, APIs, requirements, etc. Although there are many tools that focus on helping programmers find that knowledge (e.g., [16, 2, 24, 31, 25, 34]), there are surprisingly few that help programmers leverage that knowledge after it is discovered.

It is well known that programmers do not like to comment their code (for example, to add in "design rationale" for why the code is the way it is) or do extra work that they do not envision as being of immediate benefit to their main task of getting code to work [30]. However, recent research (e.g., [12, 6, 11, 13, 3, 1, 5, 8]) in other domains have identified needs and mechanisms relating to people performing complex cognitive tasks on the Internet, which have a high benefit and low cost for the individual as well as providing benefits for others later. For example, people spend a significant amount of time and effort trying to capture information and save it in structured ways. Wikipedia lists more than 20 commercial social bookmarking systems in which users provide keyword tags for web pages; in 2008, Delicious alone reported more than 5.3 million users and 180 million unique bookmarked URLs. For note-taking and organizational tools, Wikipedia lists more than 60 systems; Evernote, one of the more popular, claims more than 200 million users. Research has demonstrated the usefulness of cues from others to help an individual find what they want. Such approaches range from social navigation to social filtering to social bookmarking to social search [7, 19, 17, 33, 20, 18]. Our preliminary findings show that programmers similarly are often also collecting and organizing information for themselves, and programmers are often willing to help others when there is a clear benefit, such as answering Stack Overflow questions. We propose to leverage these trends to make it easier for programmers to collect, navigate, organize, and share knowledge. In particular, the specific research questions of our proposed research include:

- After a programmer develops knowledge for themselves, what about what they have done is generalizable for others?
- How do we capture that in a sustainable and incentivized way that will have a low cost and high perceived benefit for the initial programmer?
- How can we combine the knowledge from multiple programmers who are investigating related issues so that it will be useful for later programmers?
- How can we extract design rationale, discovered constraints, requirements and features, facts about code and APIs, and other useful information from the accumulated knowledge?
- How can we organize and present the accumulated knowledge in a way that would help the initial and later programmers?
- Where can such knowledge be surfaced so that programmers will encounter it when it is useful to them?

Our approach to start answering these research questions

will elicit theories and practices in cognitive psychology, human-computer interaction (HCI), and software engineering, and will include both new studies and novel tools. The studies will help better understand programmers' knowledge needs and activities. The tools will include extensions for browsers, text editors and integrated development environments (IDEs) to help programmers gather ("forage" [22]), navigate, organize, and share that knowledge. We envision that programmers will be able to generate and discover the knowledge they need at an accelerated pace with our supporting tools acting as a programming copilot.

Preliminary Pilot Studies

In preparation for the research and to better motivate the problem, we conducted retrospective walkthroughs with 16 experienced programmers about a complex programming task they had recently done as part of their normal programming. The goal was to identify common activities and leverage points related to the cost structure of sensemaking. While significant prior work has examined the sensemaking process [26, 4, 14, 6, 13, 12, 23], including in the context of programming [10, 9, 32], our focus here was on identifying common activities and their key costs and benefits that could inform tools to help the initial programmer while at the same time capturing their cognitive work to benefit others.

Participants engaged in a variety of tasks involving significant sensemaking activities, ranging from learning the React.js library for JavaScript (reactjs.org) to issuing bug reports, and used a variety of tools to annotate and save the information they encountered, including browser tabs and bookmarks, note-taking tools such as Workflowy, Notes, Evernote, Google Docs, or even building their own custom websites. Although the specific information needs varied across tasks, they generally fell into the two high-level sensemaking categories of *gathering* information and organizing that information into useful models to take action [26, 4, 23]. Interestingly, as a result of using a web browser to gather this information and trying to keep track of it, we also noticed a need to navigate and keep track of the various branches of options being considered, with participants encountering and queuing up possibilities to consider later [21, 29]. The types of tasks we observed programmers doing to try to accomplish these activities included:

Deciding on a framework, API, approach, pattern, or code example to use. Prior work has shown that 34% of a programmer's search sessions are aimed at finding or learning about an API [27], with each option having different strengths or limitations depending on the programmer's goals and constraints (e.g., existing coding stack, style preferences, level of expertise, etc.). We observed two participants at the beginning stages of a project, both having trouble deciding what framework or API to use. One programmer noted difficulties with foraging for information: *"The hardest part is figuring out if an API actually has the functionality I'm looking for."* Programmers often organized the information they collected to foster comparisons. One participant stated, *"Because there are so many tools [for web development], you almost want to read something that tells you, React is a good idea. Even though it's hard at first, this is why it's powerful."* They often started following one lead but switched to another when they ran into difficulties, suggesting the need for support for branching and backtracking navigation.

Learning the structure of an unfamiliar framework, API, or unfamiliar code. A majority of our interviewees cited learning unfamiliar code as a significant challenge, including new concepts (e.g., "Promise" in next-generation JavaScript) or terms (e.g., "event-driven"). One participant stated, *"A lot*

Sensemaking activity	**Task types involving sensemaking**		
	Deciding *on a framework, API, approach, pattern, or code example to use*	***Learning*** *the structure of an unfamiliar framework / API; understanding unfamiliar code*	***Implementing*** *a specific feature/functionality*
Foraging	Identifying pros and cons, important constraints, and contexts of use	Annotating unfamiliar concepts or code; capturing useful explanations	Marking pages where code was copied; annotating how it was adapted
Navigating	Keeping track of and switching among different options	Supporting deep diving into explanations and resuming main thread	Queuing potential options to implement and switching among them
Organizing	Building a comparison table of functionality	Filtering foraged tutorials and examples by users' goals	Visualizing trees of potential options
Sharing	Exposing design decisions as design rationale for other programmers	Exposing best practices where they are relevant (e.g., in code or on relevant search pages)	Shortcutting bad "rabbit holes"; showing different styles of solutions

Table 1: Design space we plan to explore, with example opportunities in each cell.

of times I was looking at the [React] tutorial, and I didn't understand what it was doing or why it was useful..." This led the programmer to want an explanation on the underlying concepts. Another participant said, *"The React documentation was very example-heavy and not concept-heavy. If you weren't careful you could miss it."* This participant found a blog post including more detailed explanations. Subsearches for these concepts or terms often led to navigation challenges with using tabs to keep track of branching, drilling down, and resuming their tasks.

Implementing a specific feature/functionality. In multiple interviews, programmers cited having trouble implementing new code even when they conceptually understood what they wanted to accomplish, e.g., *"Conceptually, I understand what's going on... how do I [load data via an http request] again? I had done it with jQuery before, but I had never done it with straight JavaScript."*

Design Space

These initial results suggest a design space to profitably explore ways to understand and help programmers **forage**, **navigate**, and **organize** different resources they encounter in order to achieve their own programming goals, and to share the results with others. Table 1 shows a preliminary depiction of this design space, with different types of programming tasks commonly occurring in our preliminary pilot study as the columns, and different activities that people engage in while interacting with information for these tasks as the rows. Within the cells are examples of opportunities

for helping support that activity for that task. For example, when deciding on a mapping API to use, the programmer might gather important constraints (e.g., whether it supports geolocation; how much does an API call cost; whether it supports overlays) and the contexts in which it can be used (e.g., if it plays well with the React framework). We propose an important advance in the addition of the sharing row, in which our main focus is the implicit sharing where a programmer's cognitive work might be used by others later. For example, if one programmer has already gathered the constraints and contexts in which they decided to use one API over others, those constraints and contexts might be surfaced as design rationale in that user's code, or they might be surfaced if another user did a similar search or visited the same web pages as the initial programmer.

One of the key goals of our research is to make this table more complete by filling out the rows and columns and the opportunities within each cell. The top three rows will help us find leverage points to create tools that capture the work that people do during sensemaking and are perceived as being useful by themselves; the last row embodies how that captured work can be made useful for others.

Conclusion

Programming has been highly cognitively demanding. We propose to go beyond previous approaches to understand and support sensemaking in programming by focusing on the opportunities to **capture the cognitive work that programmers engage in while making sense of information in order to help them and others with similar knowledge activities**. We aim to better understand the cost structure of sensemaking in programming, and use it to build tools that **directly reduce the cost and/or increase the benefit** to the initial users in making sense of information. To do so we will engage in two primary thrusts: **empirical studies** to better understand the activities, information needs, results and cost structure of sensemaking across various types of programming-related activities, and the development and evaluation of **prototype tools** to explore the design space of supporting those activities.

REFERENCES

1. Krishna Bharat. 2000. SearchPad: explicit capture of search context to support Web search. *Computer Networks* 33, 1 (June 2000), 493–501. DOI: http://dx.doi.org/10.1016/S1389-1286(00)00047-5
2. Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. 2010. Example-centric Programming: Integrating Web Search into the Development Environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 513–522. DOI: http://dx.doi.org/10.1145/1753326.1753402
3. Joseph Chee Chang, Nathan Hahn, and Aniket Kittur. 2016. Supporting Mobile Sensemaking Through Intentionally Uncertain Highlighting. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 61–68. DOI: http://dx.doi.org/10.1145/2984511.2984538
4. Brenda Dervin. 1983. An overview of sense-making research concepts, methods, and results to date. (1983). http://faculty.washington.edu/wpratt/MEBI598/Methods/An%20Overview%20of%20Sense-Making%20Research%201983a.htm
5. Mira Dontcheva, Steven M. Drucker, Geraldine Wade, David Salesin, and Michael F. Cohen. 2006. Summarizing Personal Web Browsing Sessions. In

Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06). ACM, New York, NY, USA, 115–124. DOI: http://dx.doi.org/10.1145/1166253.1166273

6. Kristie Fisher, Scott Counts, and Aniket Kittur. 2012. Distributed Sensemaking: Improving Sensemaking by Leveraging the Efforts of Previous Users. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 247–256. DOI: http://dx.doi.org/10.1145/2207676.2207711

7. Jill Freyne, Rosta Farzan, Peter Brusilovsky, Barry Smyth, and Maurice Coyle. 2007. Collecting Community Wisdom: Integrating Social Search & Social Navigation. In *Proceedings of the 12th International Conference on Intelligent User Interfaces (IUI '07)*. ACM, New York, NY, USA, 52–61. DOI: http://dx.doi.org/10.1145/1216295.1216312

8. Matthias Geel, Timothy Church, and Moira C. Norrie. 2012. Sift: An End-user Tool for Gathering Web Content on the Go. In *Proceedings of the 2012 ACM Symposium on Document Engineering (DocEng '12)*. ACM, New York, NY, USA, 181–190. DOI: http://dx.doi.org/10.1145/2361354.2361395

9. Valentina Grigoreanu, James Brundage, Eric Bahna, Margaret Burnett, Paul ElRif, and Jeffrey Snover. 2009. Males' and Females' Script Debugging Strategies. In *International Symposium on End User Development*. Springer, Berlin, Heidelberg, 205–224. DOI: http://dx.doi.org/10.1007/978-3-642-00427-8_12

10. Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Jill Cao, Kyle Rector, and Irwin Kwan. 2012. End-user Debugging Strategies: A Sensemaking Perspective. *ACM Trans. Comput.-Hum. Interact.* 19, 1 (May 2012), 5:1–5:28. DOI: http://dx.doi.org/10.1145/2147783.2147788

11. Nathan Hahn, Joseph Chang, Ji Eun Kim, and Aniket Kittur. 2016. The Knowledge Accelerator: Big Picture Thinking in Small Pieces. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 2258–2270. DOI: http://dx.doi.org/10.1145/2858036.2858364

12. Aniket Kittur, Andrew M. Peters, Abdigani Diriye, and Michael Bove. 2014. Standing on the Schemas of Giants: Socially Augmented Information Foraging. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '14)*. ACM, New York, NY, USA, 999–1010. DOI:http://dx.doi.org/10.1145/2531602.2531644

13. Aniket Kittur, Andrew M. Peters, Abdigani Diriye, Trupti Telang, and Michael R. Bove. 2013. Costs and Benefits of Structured Information Foraging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2989–2998. DOI: http://dx.doi.org/10.1145/2470654.2481415

14. G. Klein, B. Moon, and R. R. Hoffman. 2006. Making Sense of Sensemaking 1: Alternative Perspectives. *IEEE Intelligent Systems* 21, 4 (July 2006), 70–73. DOI:http://dx.doi.org/10.1109/MIS.2006.75

15. Thomas D. LaToza and Brad A. Myers. 2010. Hard-to-answer Questions About Code. In *Evaluation and Usability of Programming Languages and Tools (PLATEAU '10)*. ACM, New York, NY, USA, 8:1–8:6. DOI:http://dx.doi.org/10.1145/1937117.1937125

16. T. D. LaToza and B. A. Myers. 2011. Visualizing call graphs. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 117–124. DOI: http://dx.doi.org/10.1109/VLHCC.2011.6070388

17. David R. Millen, Jonathan Feinberg, and Bernard Kerr. 2006. Dogear: Social Bookmarking in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 111–120. DOI: http://dx.doi.org/10.1145/1124772.1124792

18. Dan Morris, Meredith Ringel Morris, and Gina Venolia. 2008. SearchBar: A Search-centric Web History for Task Resumption and Information Re-finding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1207–1216. DOI: http://dx.doi.org/10.1145/1357054.1357242

19. Meredith Ringel Morris and Eric Horvitz. 2007. SearchTogether: An Interface for Collaborative Web Search. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 3–12. DOI: http://dx.doi.org/10.1145/1294211.1294215

20. Alan Munro, Kristina HÃČÂűÃČÂűk, and David Benyon. 1999. Footprints in the Snow. Springer, London, 1–14. http://link.springer.com/10.1007/978-1-4471-0837-5_1 DOI: 10.1007/978-1-4471-0837-5_1.

21. David Piorkowski, Austin Z. Henley, Tahmid Nabi, Scott D. Fleming, Christopher Scaffidi, and Margaret Burnett. 2016. Foraging and Navigations, Fundamentally: Developers' Predictions of Value and Cost. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. ACM, New York, NY, USA, 97–108. DOI: http://dx.doi.org/10.1145/2950290.2950302

22. Peter Pirolli and Stuart Card. 1995. Information Foraging in Information Access Environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 51–58. DOI: http://dx.doi.org/10.1145/223904.223911

23. Peter Pirolli and Stuart Card. 2005. The Sensemaking Process and Leverage Points for Analyst Technology as Identified Through Cognitive Task Analysis. In *Proceedings of International Conference on Intelligence Analysis*. http://www.phibetaiota.net/wp-content/uploads/2014/12/Sensemaking-Process-Pirolli-and-Card.pdf

24. Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. 2013. Seahawk: Stack Overflow in the IDE. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, USA, 1295–1298. DOI:http://dx.doi.org/10.1109/ICSE.2013.6606701

25. M.P. Robillard and G.C. Murphy. 2003. Automatically Inferring Concern Code from Program Investigation Activities. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.* IEEE Comput. Soc, Montreal, Que., Canada, Canada, 225–234. DOI: http://dx.doi.org/10.1109/ASE.2003.1240310

26. Daniel M. Russell, Mark J. Stefik, Peter Pirolli, and Stuart K. Card. 1993. The Cost Structure of Sensemaking. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, New York, NY, USA, 269–276. DOI: http://dx.doi.org/10.1145/169059.169209

27. Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How Developers Search for Code: A Case Study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 191–201. DOI: http://dx.doi.org/10.1145/2786805.2786855

28. Jonathan Sillito, Gail C. Murphy, and Kris De Volder. 2006. Questions Programmers Ask During Software Evolution Tasks. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '06/FSE-14)*. ACM, New York, NY, USA, 23–34. DOI: http://dx.doi.org/10.1145/1181775.1181779

29. Justin Smith, Chris Brown, and Emerson Murphy-Hill. 2017. Flower: Navigating Program Flow in the IDE. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'17)*. Raleigh, NC, USA, 19–23. http://www4.ncsu.edu/~jssmit11/Publications/VLHCC17_Flower.pdf

30. Diomidis Spinellis. 2003. Reading, Writing, and Code. *Queue* 1, 7 (Oct. 2003), 84–89. DOI: http://dx.doi.org/10.1145/957717.957782

31. J. Stylos and B.A. Myers. 2006. Mica: A Web-Search Tool for Finding API Components and Examples. In *Visual Languages and Human-Centric Computing (VL/HCC'06) (VL/HCC'06)*. IEEE, 195–202. DOI: http://dx.doi.org/10.1109/VLHCC.2006.32

32. Neeraja Subrahmaniyan, Laura Beckwith, Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Vaishnavi Narayanan, Karin Bucht, Russell Drummond, and Xiaoli Fern. 2008. Testing vs. Code Inspection vs. What else?: Male and Female End Users' Debugging Strategies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 617–626. DOI: http://dx.doi.org/10.1145/1357054.1357153

33. Michael B. Twidale, David M. Nichols, and Chris D. Paice. 1997. Browsing is a collaborative process. *Pergamon Infarmation Processing & Management* 33, 6 (Nov. 1997), 761–783. DOI: http://dx.doi.org/10.1016/S0306-4573(97)00040-X

34. Jeremy Warner and Philip J. Guo. 2017. CodePilot: Scaffolding End-to-End Collaborative Software Development for Novice Programmers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 1136–1141. DOI: http://dx.doi.org/10.1145/3025453.3025876